**CS1101: Lab 5 – Digital Logic Design with Descendant Trees**

An entire subfield of Electrical and Computer Engineering is dedicated to the design of digital logic circuits. Digital circuits are those whose values are confined to 1s and 0s rather than the entire analog spectrum (any value between 0 and 1). Digital circuits are generally constructed of components such as AND, OR, NOR, NAND, XOR, XNOR, or NOT gates. They can also include more complicated components such as multiplexors, decoders, or flip-flops. In this lab, we will be creating digital circuits using descendant trees in Scheme.

**Lab Motivation and Goals**

* Program functions to process descendant trees

**Exercises**

In order to properly construct a digital logic circuit, an electrical designer needs to know certain properties of the circuit. The information that he requires includes the name of the output signal from the circuit (string) so that it can be identified on a schematic, the logic value of the output signal from the circuit (number), the operation or component that was used to create the output signal (symbol), the strength of the output signal (symbol), the propagation delay of the output signal through the logic gate in nanoseconds (number), as well as a list of input signals that produced the output signal. It should be recognized that the logic value of signals in the circuit can only be 0 or 1. The operation or gates used to create signals should be limited to the simple gates mentioned in the introduction.

1. Develop a data definition and an example of a digital logic circuit. The logic circuit should start with a single signal (the output). The data for the input signals to the circuit should be a list of other circuits or empty if the given signal is an input to the circuit.
2. Provide a template for digital circuits.
3. Write a function **count** that consumes a circuit and a symbol representing a gate type and produces the number of those gates used to create the circuit.
4. Write a function **OR-used?** that consumes a circuit and produces a boolean that is true if OR-gates are used in the circuit and false otherwise.
5. Write a function **find-weak** that consumes a circuit and produces a list of string representing the names of all signals that are weak in the circuit.

**Everyone should be able to finish up to this point.**

1. Write a function **count-inputs** that consumes a circuit and produces the number of input signals to the circuit. A signal is an input if its input signal list is empty.
2. Write a function **all-strong?** that consumes a circuit and produces a boolean, which is true if all signals in the circuit are strong and false otherwise.
3. Write a function **reduce-delay** that consumes a circuit and a symbol representing a gate-type and produces a circuit such that all signals produced by that gate-type have their propagation delay reduced by 10%.
4. Write a function **find-signal** that consumes a circuit, a number representing a logic value, a symbol representing a signal strength, a number representing a propagation delay, and a symbol representing a gate type. It produces the name of the first signal in the circuit that matches all the input values and ‘none if such a signal does not exist.

**Wow! You’re moving fast. The last two problems are slightly more difficult, but give them a try!**

1. Write a function **new-input** that consumes a symbol representing a gate-type, a number representing a logic value, a symbol representing a trace name, and a circuit. It produces the same circuit; however, the first signal in the circuit with the gate-type input to the function will have a new input signal added to its list of input signals. This new input signal will have a logic value equal to the logic value input to the function; a trace name equal to the trace name input to the function; gate-type, strength, and propagation delay equal to the signal that it created; and an empty list-of-signals since it is an input. (Note: The tricky part of this function is to make sure that the new signal is only added to one input signal list rather than multiple. Keep this in mind while creating the function.)
2. Write a function **count-errors** that consumes a circuit and produces a number representing the number of output signals from the circuit with the wrong logic value based on the operation used to create that signal and the logic values of the inputs signals to that signal. You are only responsible for counting errors produced by AND and OR gates.

As a digital logic reminder, an AND gate should only produce a 1 when all the inputs to the gate are 1; otherwise, it will produce a 0. An OR gate should only produce a 0 when all the inputs to the gate are 0; otherwise, it will produce a 1.

(Hint: You may want to create helper functions that will determine if an error has occurred in the circuit. Functions that determine if a list of input signals contains a signal with a logic value of 1 or 0 may also help.) (Note: This function is much harder than the others created in this lab. Therefore, if you can’t get it, don’t worry, but give it a try anyway.)